Exercises Pre-Master Bridging Program

Python

Session: DATES

Exam Base in KFSC – Riyadh

Instructor names: Dr. Rama Ayoub, Dr. Roberto Nuca, Dr. Fabio Credali

Student Name in Arabic:

Student Name in English:

Student National ID No.:

Email Address registered:

1 Getting Started

1. Install Python and Verify Installation

Install Python on your machine. After installation, verify that everything is working correctly by opening a terminal (or command prompt) and typing `python` (or `python3`). You should see the Python version displayed.

2. Print "Hello World"

Write a Python program that prints `"Hello world"` using the `print(...)` function.

3. Save Program to a File

Save the content of your program in a file with the extension `.py`. For example, save the program from Exercise 2 in a file called `hello_world.py`.

4. Assign a Value to a Variable

Write a Python statement that creates a variable `x` and assigns it the value `10`.

5. Change the Value of a Variable

Write two lines of code. In the first, assign the value `5` to `y`, and in the second, change `y`'s value to `12`. Then, print both values.

6. Read and Print a Sentence

Read a sentence as input and print it using the functions `input(...)` and `print(...)`.

7. Print Your Name

Read your name as input and save it to a variable. Then, print your name preceded by `"Hello :"`.

8. Perform Addition of Two Numbers

Write three lines of code:

- Assign `7` to `a`.
- Assign `3` to `b`.
- Add `a` and `b` together, and store the result in `result`.

9. Add Inline Comments

Write inline comments (using `#`) to describe the content of your program.

10. Swapping Two Variables

Swapping Variables:

- Assign any number to `x` and another number to `y`.
- Afterward, swap the values of `x` and `y`.

11. Perform a Mathematical Calculation

Assign the values `8` to `x`, `2` to `y`, and `3` to `z`. Then, compute the result of `(x + y) * z` and store the result in a variable `final_result`.

12. Descriptive Variable Names

Assign the value `50` to a variable `score`. Then, assign `3` to a variable `attempts`. Finally, print a sentence that says, `"Your score is <score> and you have <attempts> attempts left."`

2 Data Types

1. Print a String

Create a variable `greeting` that stores the string `"Hello, Python!"` and print it.

2. Concatenate Strings

Create two variables `first_name` and `last_name` that store your first and last name as strings. Concatenate them with a space in between and print the full name.

3. Integer Operations

Create two variables `a` and `b` and assign them integer values. Calculate and print their sum, difference, product, and quotient.

4. Convert String to Integer

Create a variable `num_str` that contains a string representation of an integer (e.g., `"42"`). Convert it to an integer and print the result.

5. Float Operations

Create two variables `x` and `y` and assign them float values. Calculate and print the sum, difference, and product of `x` and `y`.

6. Boolean Values

Create two boolean variables `is_raining` and `has_umbrella`. Set `is_raining` to `True` and `has_umbrella` to `False`. Print the logical AND of both variables.

7. Check Data Type

Create a variable `value` that stores the number `7.5`. Use the `type()` function to print the data type of `value`.

8. Type Casting

Create a float variable `pi` with the value `3.14159`. Convert it to an integer and print the result.

9. Float Division

Create two variables `a` and `b` and assign them float values (e.g., `a = 9.0` and `b = 4.0`). Calculate and print the result of dividing `a` by `b`.

10. String Length and Concatenation

Create a variable `word1` with the value `"Python"` and a variable `word2` with the value `"rocks"`. First, print the length of `word1` using the `len()` function. Then, concatenate `word1` and `word2` with a space between them and print the result.

11. Integer Division and Remainder

Create two integer variables `a` and `b` and assign them values (e.g., `a = 15` and `b = 4`). First, print the result of integer division (use `//`) of `a` by `b`. Then, print the remainder when `a` is divided by `b` using the modulus operator `%`.

12. Floating Point Precision

Create a float variable `pi` with the value `3.14159265359`. Use the `round()` function to round `pi` to two decimal places and print the result. Then, print the result of dividing `pi` by 2 and rounding to three decimal places.

13. Boolean Logic with Comparison Operators

Create two variables, `x` and `y`, with integer values (e.g., `x = 5` and `y = 10`). First, use a comparison operator to check if `x` is greater than `y` and print the result. Then, check if `x` is less than or equal to `y` and print the result. Finally, use the logical `and` operator to check if `x` is greater than 0 *and* less than 10, and print the result.

14. Type Conversion Between Data Types

Create a string variable `num_str` with the value `"20"`. Convert `num_str` to an integer and store it in a variable `num_int`. Then, convert `num_int` to a float and store it in a variable `num_float`. Print both `num_int` and `num_float`.

3 Working with lists

1. Create a List of Numbers

Create a list containing the numbers 1, 2, 3, and 4. Print the list.

2. Access Elements of a List

Access and print the first element from the list `[10, 20, 30, 40]`.

3. Change an Element in a List

Change the second element of the list `[1, 2, 3, 4]` to 10 and print the list.

4. Add an Element to the End of a List

Add the number 5 to the list `[1, 2, 3, 4]` and print the updated list.

5. Remove an Element from a List

Remove the number 3 from the list `[1, 2, 3, 4]` and print the list.

6. Concatenate Two Lists

Create two lists, `[1, 2]` and `[3, 4]`, and concatenate them into a single list. Print the result.

7. Find the Length of a List

Find and print the length of the list `[10, 20, 30]`.

8. Check if an Element Exists in a List

Check if the number 10 exists in the list `[1, 2, 3, 4, 5]`. Print `True` if it exists, `False` otherwise.

9. Create a List with Mixed Data Types

Create a list containing a string, an integer, and a float, for example `["Hello", 10, 3.14]`. Print the list.

10. Access the Last Element of a List

Access and print the last element of the list `[5, 10, 15, 20]`.

11. Slice a List

Create a list `[1, 2, 3, 4, 5]` and slice it to get the first three elements. Print the sliced list.

12. Repeat a List

Create a list `[1, 2]` and repeat it three times. Print the result.

13. Nested List

Create a list that contains another list, for example `[[1, 2], [3, 4]]`. Print the nested list.

14. Get a Sublist from a Nested List

Given the list `[[1, 2], [3, 4], [5, 6]]`, print the second sublist `[3, 4]`.

15. Combine a String and a List

Create a list `[10, 20]` and combine it with the string `"Numbers:"`. Print the result as `"Numbers:  10, 20"`.

4 If Statements

1. Check if a Number is Positive or Negative

Given the number `5`, check if it is positive or negative using an if-else statement and print the result.

2. Check if a Number is Even or Odd

Given the number `4`, check if it is even or odd using an if-else statement and print the result.

3. Check if a Number is Zero

Given the number `0`, check if it is zero and print the appropriate message.

4. Check if a Number is Divisible by 3 and 5

Given the number `15`, check if it is divisible by both 3 and 5 and print the result.

5. Compare Two Numbers

Given the numbers `a = 10` and `b = 20`, compare them and print which one is larger.

6. Check if a Character is a Vowel

Given the character `'a'`, check if it is a vowel (a, e, i, o, u) and print the result.

7. Check if a Year is a Leap Year

Given the year `2024`, check if it is a leap year and print the result. A leap year is exactly divisible by 4, except if it is divisible by 100.

8. Check the Largest of Three Numbers

Given the numbers `a = 10`, `b = 20`, and `c = 15`, check which one is the largest.

9. Check if a Number is Within a Range

Given the number `8`, check if it is between 5 and 10 (inclusive). Print the result.

10. Check if a Number is Positive, Negative, or Zero

Given the number `-3`, check if it is positive, negative, or zero and print the result.

11. Check if a Number is a Multiple of 4 or 6

Given the number `24`, check if it is a multiple of 4 or 6.

12. Check if a String is Empty

Given the string `"Hello"`, check if the string is empty.

13. Check if a Person is Eligible to Drive

Given the age `age = 18`, check if the person is eligible to drive (must be 18 or older).

14. Check if a Number is Positive and Less Than 100

Given the number `50`, check if it is positive and less than 100.

15. Check if a Character is a Digit

Given the character `'7'`, check if it is a digit.

5 Dictionaries

1. Create a Dictionary

Create a dictionary with the following key-value pairs: `"name": "Alice", "age": 25`. Print the dictionary. In Python, dictionaries are created using curly braces `{}`. You can define key-value pairs, where the key is a unique identifier, and the value is the corresponding information.

2. Access a Value by Key

Given the dictionary `person = {"name": "Alice", "age": 25}`, access and print the value associated with the key `"name"`. To access a value from a dictionary, use square brackets and specify the key inside the brackets.

3. Add a New Key-Value Pair

Given the dictionary `person = {"name": "Alice", "age": 25}`, add a new key `"city"` with the value `"New York"` and print the updated dictionary. To add a new key-value pair, simply assign a value to a new key using square brackets.

4. Update the Value of an Existing Key

Given the dictionary `person = {"name": "Alice", "age": 25}`, update the value of the key `"age"` to 26 and print the updated dictionary. To update the value of an existing key, assign a new value to that key.

5. Check if a Key Exists in a Dictionary

Given the dictionary `person = {"name": "Alice", "age": 25}`, check if the key `"age"` exists in the dictionary and print `True` if it does, `False` otherwise. To check if a key exists in a dictionary, you can use the `in` keyword.

6. Remove a Key-Value Pair

Given the dictionary `person = {"name": "Alice", "age": 25}`, remove the key `"age"` and print the updated dictionary. To remove a key-value pair, use the `del` statement followed by the key in square brackets.

7. Get a Value Using `get()`

Given the dictionary `person = {"name": "Alice", "age": 25}`, use the `get()` method to access the value associated with the key `"name"` and print it. The `get()` method allows you to safely retrieve the value for a key. If the key doesn't exist, it returns `None` by default.

8. Default Value with `get()`

Given the dictionary `person = {"name": "Alice", "age": 25}`, use the `get()` method to access the value associated with the key `"city"`. If the key does not exist, return `"Unknown"` and print it. The `get()` method can also accept a second argument, which will be returned if the key is not found.

9. Get the Keys of a Dictionary

Given the dictionary `person = {"name": "Alice", "age": 25}`, get and print all the keys in the dictionary. To get all the keys in a dictionary, you can use the `keys()` method.

10. Get the Values of a Dictionary

Given the dictionary `person = {"name": "Alice", "age": 25}`, get and print all the values in the dictionary. To get all the values in a dictionary, use the `values()` method.

11. Get the Items of a Dictionary

Given the dictionary `person = {"name": "Alice", "age": 25}`, get and print all the key-value pairs (items) in the dictionary. To get all key-value pairs as tuples, use the `items()` method.

12. Clear All Key-Value Pairs

Given the dictionary `person = {"name": "Alice", "age": 25}`, clear all key-value pairs from the dictionary and print the empty dictionary. To remove all items from the dictionary, use the `clear()` method.

13. Merge Two Dictionaries

Given the dictionaries `person1 = {"name": "Alice", "age": 25}` and `person2 = {"city": "New York", "job": "Engineer"}`, merge them into a new dictionary and print the result. To merge two dictionaries, you can use the unpacking operator `**`.

14. Nested Dictionary

Given the dictionary `person = {"name": "Alice", "address": {"city": "New York", "zip": "10001"}}`, access and print the value of the key `"city"` inside the nested dictionary. To access a value inside a nested dictionary, you can use multiple square brackets.

15. Check if a Dictionary is Empty

Given the dictionary `person = {"name": "Alice", "age": 25}`, check if the dictionary is empty and print `True` if it is empty, `False` otherwise. You can check if a dictionary is empty by using the `not` operator. An empty dictionary will evaluate to `False`.

6 For and While Loops

1. Print Numbers from 1 to 5 using a `for` Loop

Write a `for` loop that prints the numbers from 1 to 5. The `for` loop can iterate over a range of numbers. Use `range()` to generate the numbers.

2. Print Elements of a List using a `for` Loop

Given the list `numbers = [2, 4, 6, 8, 10]`, write a `for` loop to print each element of the list. A `for` loop can also iterate over elements of a list.

3. Print Numbers from 1 to 10 using a `while` Loop

Write a `while` loop that prints the numbers from 1 to 10. A `while` loop continues as long as a condition is `True`.

4. Print Sum of Numbers from 1 to 5 using a `for` Loop

Write a `for` loop that calculates and prints the sum of the numbers from 1 to 5. Use a variable to accumulate the sum as you iterate over the numbers.

5. Print Even Numbers from 1 to 10 using a `for` Loop

Write a `for` loop to print all even numbers from 1 to 10. Use `range()` with a step argument to get only even numbers.

6. Count Down from 5 to 1 using a `while` Loop

Write a `while` loop that counts down from 5 to 1. Decrement the variable inside the loop to decrease the count.

7. Print the First 5 Multiples of 3 using a `for` Loop

Write a `for` loop to print the first 5 multiples of 3. Use `range()` to iterate through numbers and multiply each by 3.

8. Print the Sum of Numbers in a List using a `for` Loop

Given the list `numbers = [1, 2, 3, 4, 5]`, write a `for` loop that calculates and prints the sum of the numbers. The `for` loop can be used to iterate over the elements and accumulate the total.

9. Print the Squares of Numbers from 1 to 5 using a `for` Loop

Write a `for` loop that prints the squares of the numbers from 1 to 5. Use the exponentiation operator (`**`) to calculate the square of each number.

10. Print Odd Numbers from 1 to 9 using a `while` Loop

Write a `while` loop that prints all odd numbers from 1 to 9. You can check if a number is odd using the modulo operator (`%`).

11. Print Numbers from 10 to 1 using a `while` Loop

Write a `while` loop that prints the numbers from 10 to 1 in reverse order. You can decrement the counter to print the numbers in reverse.

12. Print Numbers Divisible by 3 from 1 to 15 using a `for` Loop

Write a `for` loop that prints all numbers divisible by 3 from 1 to 15. Use the modulo operator (`%`) to check divisibility by 3.

13. Print the Length of a List using a `for` Loop

Given the list `fruits = ["apple", "banana", "cherry"]`, write a `for` loop to count and print the length of the list. You can count the length by iterating over each element of the list.

14. Find the Largest Number in a List using a `for` Loop

Given the list `numbers = [10, 15, 20, 25, 30]`, write a `for` loop to find and print the largest number. You can compare each number as you iterate through the list.

15. Break the Loop if Number is 5

Write a `for` loop that prints the numbers from 1 to 10, but breaks the loop when the number is 5. Use the `break` statement to exit the loop early.

16. Using `pass` in an Empty `if` Block

Write a program that checks if a number is positive, negative, or zero. Use the `pass` statement inside the `if` block when the number is zero. The `pass` statement can be used in a block where you haven't implemented any logic yet.

17. Skipping Even Numbers in a `for` Loop

Write a `for` loop that iterates through the numbers from 1 to 10. Use the `continue` statement to skip printing even numbers. The `continue` statement can be used to skip the current iteration of the loop when a condition is met.

18⋆. Sum of Digits Using For Loop

Write a Python program that takes an integer n as input, and uses a `for` loop to compute the sum of its digits. The program should print the sum.

19⋆. Reverse a String Using For Loop

Write a Python program that takes a string as input and uses a `for` loop to reverse the string. The program should print the reversed string.

20⋆. Find the Longest Word in a Sentence Using For Loop

Write a Python program that takes a sentence as input and uses a `for` loop to find the longest word in the sentence. The program should print the longest word.

7 Functions

1. Define a Simple Function

Define a function `greet()` that prints `"Hello, World!"` when called. A simple function can be defined using the `def` keyword.

2. Function with Parameters

Define a function `greet(name)` that takes a parameter `name` and prints `"Hello, {name}!"` when called. A function can accept parameters and use them inside the body.

3. Function with Return Value

Define a function `add(a, b)` that takes two numbers `a` and `b` and returns their sum. Functions can return values using the `return` keyword.

4. Function with Default Argument

Define a function `greet(name="Guest")` that greets a user with the provided name, or with `"Guest"` if no name is provided. Functions can have default arguments that are used when no value is passed.

5. Function with Variable Number of Arguments

Define a function `sum_numbers(*args)` that takes any number of arguments and returns their sum. You can use the `*args` syntax to accept a variable number of arguments.

6. Function Returning Another Function

Define a function `multiply_by(x)` that returns a function which multiplies its input by `x`. Functions can return other functions, which can be called later.

7. Lambda Function for Addition

Create a lambda function `add_lambda` that takes two arguments and returns their sum. Lambda functions are anonymous functions defined using the `lambda` keyword.

8. Lambda Function for Squaring a Number

Create a lambda function `square` that takes one argument and returns its square. Lambda functions are typically used for short operations like mathematical calculations.

9. Recursive Function to Calculate Factorial

Define a recursive function `factorial(n)` that calculates the factorial of `n`. A recursive function calls itself with modified arguments until a base case is met.

10. Function with Multiple Return Values

Define a function `min_max(a, b)` that returns both the minimum and maximum of two numbers. A function can return multiple values as a tuple.

11. Using `map` with a Function

Define a function `square(x)` and use the `map()` function to apply it to a list `[1, 2, 3, 4]`. The `map()` function applies a given function to all items in an iterable.

12. Using `filter` with a Function

Define a function `is_even(x)` and use the `filter()` function to filter out odd numbers from a list `[1, 2, 3, 4, 5]`. The `filter()` function filters elements from an iterable based on a condition defined by a function.

13. Function to Check if a Number is Prime

Define a function `is_prime(n)` that checks if a number `n` is prime. This function checks divisibility for numbers up to `sqrt(n)`.

14. Function to Count Vowels in a String

Define a function `count_vowels(s)` that counts the number of vowels in the string `s`. You can iterate through the string and check if each character is a vowel.

15. Function to Reverse a String

Define a function `reverse_string(s)` that takes a string `s` and returns it reversed. You can reverse a string by slicing it with `s[::-1]`.

16. Lambda Function with Multiple Expressions

Create a lambda function `process` that accepts a number and performs two operations: adds 5, then multiplies by 2. Return the final result. In advanced lambda functions, you can include multiple expressions by using parentheses.

17. Higher-order Function: Function that Returns a Function

Define a function `power(n)` that returns a function which raises its argument to the power of `n`. This is a higher-order function, as it returns another function.

18. Function that Takes Functions as Arguments

Define a function `apply_function(f, x)` that takes a function `f` and an argument `x`, and returns the result of applying `f` to `x`.

19. Function with Keyword Arguments

Define a function `create_greeting(name, greeting="Hello")` that takes a name and an optional greeting, and prints a greeting message.

20. Function with Variable-Length Keyword Arguments

Define a function `print_info(*kwargs)` that prints out all key-value pairs from the `kwargs` dictionary. You can use `**kwargs` to accept any number of keyword arguments.

21*. Function for Finding Common Elements in Two Lists

Write a Python function that takes two lists as input and returns a list of elements that are common in both lists. The function should not use set operations but should use a loop to check for common elements.

22*. Function to Find the Number of Occurrences of Each Character in a String

Write a Python function that counts the occurrences of each character in a given string and returns a dictionary where the keys are the characters and the values are their respective counts.

23*. Function to Remove Duplicates from a List

Write a Python function `remove_duplicates(lst)` that removes all duplicate values from a list. The function should return a new list that contains only unique elements while preserving the original order of the list. The function should not use built-in functions such as `set()` or `dict()`.

8 Working with Files

1. Writing to a Text File

Write a program that creates a new text file called `file.txt` and writes the string `"Hello, World!"` into it. You can use the `open()` function with the `"w"` mode to create and write to a file.

2. Reading from a Text File

Write a program that opens the file `file.txt` and prints its contents. Use the `open()` function with the `"r"` mode to read from the file.

3. Writing Multiple Lines to a File

Write a program that writes the following lines to a file called `myfile.txt`:

- Line 1
- Line 2
- Line 3

You can write multiple lines to a file using `writelines()` or by calling `write()` for each line.

4. Reading Lines from a File

Write a program that reads and prints each line from the file `myfile.txt`. You can read lines individually using `readlines()`.

5. Appending to a File

Write a program that appends the string `"Goodbye, World!"` to the existing file `file.txt`. To append to a file, use the `"a"` mode in `open()`.

6. Checking if a File Exists

Write a program that checks if the file `myfile.txt` exists. If it does, print `"File exists"`, otherwise print `"File does not exist"`. You can use the `os.path.exists()` method to check for file existence.

7. Counting the Number of Lines in a File

Write a program that counts the number of lines in the file `myfile.txt`. You can use `readlines()` to get all lines and then count them.

8. Replacing Text in a File

Write a program that replaces all occurrences of the word `"Hello"` with `"Hi"` in the file `file.txt`. Read the content of the file, modify it, and then write it back to the file.

9. Writing Data to a CSV File

Write a program that creates a CSV file named `data.csv` and writes the following data:

- Name, Age
- Alice, 25
- Bob, 30

Use the `csv` module to write structured data to a CSV file.

10. Reading and Processing a CSV File

Write a program that reads `data.csv` and prints each row of the file. Use the `csv` module to read structured data from a CSV file.

11. Writing to a Binary File

Write a program that creates a binary file named `data.bin` and writes the integer `1234` as binary data. To write binary data, use the `wb` mode in the `open()` function.

12. Reading from a Binary File

Write a program that reads the binary data from the file `data.bin` and prints it as a string. Use the `rb` mode to read binary data from the file.

13. Appending to a Binary File

Write a program that appends the integer `5678` (as binary) to the existing binary file `data.bin`. You can append to a binary file using the `ab` mode in the `open()` function.

14. Saving and Loading a List and Dictionary with Binary Files

Write a program that saves a list `[1, 2, 3, 4, 5]` and a dictionary `{"name": "Alice", "age": 30}` to a binary file called `data.pkl`, and then loads them back into variables. Use the `pickle` module to serialize the objects and write them to a binary file, and then read them back and deserialize.

15⋆. Function to Write a Log File for Errors

Write a Python function `log_error(message, log_file)` that logs error messages to a specified log file. The function should append the message to the log file with a timestamp, in the format: `YYYY-MM-DD HH:MM:SS - Error: message`. If the log file does not exist, it should be created.

- The function logs error messages to a file, including a timestamp in the format `YYYY-MM-DD HH:MM:SS`. It appends messages to the file if it already exists, and if the file doesn't exist, it creates one automatically.
- You can read current date and time with `datetime.now().strftime('%Y-%m-%d %H:%M:%S')`.

16⋆. Calculate Average Score from a CSV File

Write a Python function `calculate_average(input_file, output_file)` that:

- Reads a CSV file (`input_file`) with two columns: `name` and `score`.
- The function should calculate the average score of all students.
- Write the average score to a new file (`output_file`).

The output file should contain:

- The first line: `Average Score: <average_score>`.

The function should print an error message if the file does not exist or if there are invalid scores in the data.

9 Introduction to Numpy

1. Create a NumPy Array

Create a NumPy array containing the numbers from 1 to 10.

2. Array of Zeros

Create a NumPy array of 5 zeros.

3. Array of Ones

Create a NumPy array of 3 ones.

4. Array with a Range of Values

Create a NumPy array of even numbers from 2 to 20.

5. Reshaping an Array

Create a 1D array with 12 elements and reshape it into a 3x4 matrix.

6. Array Indexing

Create a NumPy array of 5 numbers and access the third element.

7. Slicing an Array

Create a NumPy array of 10 numbers and slice it to get the first 5 elements.

8. Array with Random Values

Create a 3x3 NumPy array of random integers between 1 and 10.

9. Element-wise Operations

Create two NumPy arrays and add them element-wise.

10. Array Broadcasting

Create two NumPy arrays, one 1D array with 3 elements and another 2D array, and add them together using broadcasting.

11. Array Mathematical Operations

Create a NumPy array and compute the sum, product, and mean of the elements.

12. Transposing a Matrix

Create a 2x3 matrix and transpose it.

13. Matrix Multiplication

Create two NumPy matrices and perform matrix multiplication.

14. Finding Maximum and Minimum Values

Create a NumPy array and find the maximum and minimum values in it.

15. Flattening a 2D Array

Create a 2D NumPy array and flatten it into a 1D array.

16. Checking for NaN Values

Create a NumPy array with a NaN value and check if it contains NaN using `np.isnan()`.

17. Finding Indices of Non-zero Elements

Create a NumPy array with some zeros and use `np.nonzero()` to find the indices of non-zero elements.

18. Stacking Arrays Horizontally

Create two NumPy arrays and stack them horizontally.

19. Stacking Arrays Vertically

Create two NumPy arrays and stack them vertically.

20. Sorting an Array

Create a NumPy array and sort it in ascending order.

21*. Array Filtering

Write a Python function `filter_array(arr, threshold)` that:

- Takes a 1D NumPy array `arr` and a `threshold` value.
- Returns a new array containing only the elements from `arr` that are greater than the threshold.

22⋆. Advanced Indexing with Multi-dimensional Arrays

Write a Python function `extract_and_modify(arr)` that:

- Takes a 3D NumPy array `arr` of shape (x, y, z).
- Extracts and returns:
  - The first 2D slice of the array (along the first axis).
  - A sub-array consisting of the last 2 rows and columns of the last 2D slice.
- Modifies the second slice by setting all elements greater than 0.5 to 1 and returns the modified array.

23⋆. Masking and Conditional Operations

Write a Python function `mask_and_apply(arr)` that:

- Takes a 2D NumPy array `arr` of shape (m, n).
- Creates a boolean mask where all elements greater than 0.7 are marked as `True` and others as `False`.
- Applies this mask to set all values greater than 0.7 to their square and the others to 0, and returns the modified array.

24⋆. Efficient Aggregation with Axis Manipulation

Write a Python function `aggregate_and_normalize(arr)` that:

- Takes a 2D NumPy array `arr` of shape (m, n).
- Computes the sum of each column and the mean of each row using `np.sum()` and `np.mean()`.
- Normalizes the array such that each column has a sum of 1 by dividing each column by its sum.
- Returns the sum of each column, the mean of each row, and the normalized array.

10 Classes

1. Define a Simple Class

Define a class called `Person` with an attribute `name` and a method `greet()` that prints a greeting with the person's name.

2. Create an Object from a Class

Create an object of the `Person` class and call its `greet()` method.

3. Add More Attributes to a Class

Add an attribute `age` to the `Person` class and print both the name and age in the `greet()` method.

4. Modify an Attribute of an Object

Create a `Person` object and modify its `age` attribute. Print the updated information.

5. Class with Multiple Methods

Define a class `Car` with methods `start()` and `stop()`. Each method should print a message indicating whether the car is starting or stopping.

6. Use of `__str__` Method

Define a class `Book` with attributes `title` and `author`. Implement the `__str__()` method to return a string representation of the book.

7. Class Inheritance

Create a class `Animal` with a method `make_sound()`. Create a subclass `Dog` that overrides `make_sound()` to print "Woof".

8. Using Super()

In the previous example, call the `make_sound()` method from the parent class `Animal` within the `Dog` class using the `super()` function.

9. Static Methods in a Class

Define a class `MathOperations` with a static method `add()` that takes two arguments and returns their sum. Call this static method without creating an object.

10. Class Variables

Create a class `Counter` with a class variable `count` that increments every time a new object is created. Print the value of `count` after creating several objects.

11*: Bank Account Class

Write a Python class `BankAccount` that simulates a bank account. The class should have:

- An instance variable `balance` initialized to 0.
- A method `deposit(amount)` that increases the balance by `amount`.
- A method `withdraw(amount)` that decreases the balance by `amount`, provided there are sufficient funds.
- A method `get_balance()` that returns the current balance.

Include error handling for withdrawals that exceed the balance.

12*. Car Class with Getter and Setter Methods

Create a Python class `Car` that models a car's properties and behaviors. The class should have:

- Two instance variables: `make` and `model` for the car's brand and model.
- A private instance variable `_fuel_level` to represent the amount of fuel in the car (in percentage).
- A method `drive(distance)` that reduces the fuel level by `distance / 10`, simulating fuel consumption.
- Getter and setter methods for the `_fuel_level`, ensuring it cannot be set to a negative value.
- A method `get_car_info()` that returns the car's brand, model, and current fuel level.